IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

Applicant	: Mark D. Conover	Docket no. 2134
Serial no	: 09/168,644	
Filed	: October 8, 1998	
For	: ENCODING A STILL IMAGE INTO COMPRESSED VIDEO	
Art Unit	: 2613	Examiner: Richard J. Lee

DECLARATION OF MARK D. CONOVER INCLUDING A DECLARATION UNDER 37 C.F.R. § 1.131

I, MARK D. CONOVER, declare that:

1. I am 52 years old and reside at 10721 Wunderlich Drive, Cupertino, California 95014.

2. In 1972, I received a Bachelors of Science degree in computer science from Carnegie Mellon University. Subsequently, in 1975, I received a Masters degree in computer science from Rensselaer Polytechnic Institute

3. I am a co-founder of Pixel Tools Corporation ("Pixel Tools"), the assignee of the patent application identified above, and presently hold the position of Chief Technical Officer in the corporation.

4. For at least the preceding 8 years I have been employed as a consultant computer programmer designing and implementing various products for various different companies that are based upon several of the various different Moving Picture Experts Group ("MPEG") standards.

5. I am the applicant identified above for this patent application.

6. I have reviewed the Examiner's Action that issued for this patent application dated February 12, 2002, particularly United States Patent no. 6,324,217 entitled "Method and Apparatus for Producing an Information Stream Having Still Images" which issued November 27, 2001, on an application filed July 8, 1998, by Donald F. Gordon ("the Gordon patent").

7. In reviewing the Gordon patent I find that it fails to disclose:

a. rate control which is essential for certain MPEG applications;

b. anything about the Gordon patent's NULL-P frame except that it has a zero motion vector component;"[1]

c. that the encoded I-Frame must be shorter than the buffer in the MPEG decoder; and

d. anything about visual pulsing of a decoded image.

8. As demonstrated by texts that appear at various places in my present patent application including the computer program texts that appear on pages 27 and 28, a NULL P frame. which when included in an elementary MPEG bit stream produces images which don't pulse visually after decoding, is far more complicated that merely having "a zero motion vector component" as disclosed in the Gordon patent.

---

[1] See the Gordon patent in col. 6 at line 14.

9. The Gordon patent, in col. 6 beginning at lines 15, expressly states that:

> the GOP bit budget is utilized by 14 P-frames (approximately 38 bytes each), additional overhead information (a relatively constant amount) and a single I-frame. Thus, the quantization parameters are calculated based upon a maximal bit budget utilization for encoding the I-frame.

The Gordon patent reiterates the preceding assertion in two successive paragraphs which begin in column 7 at line 26.

10. As disclosed in the pending patent application in a text that begins on page 14 at line 13 that:

> parameters supplied to the video encoder 38 are preferably chosen so the amount of data produced for the I frame 54 approaches, but remains less than, the storage capacity of the buffer memory included in the decoder.

11. MPEG-1 decoders typically include a buffer that stores 40,960 bytes of a compressed video data. MPEG-2 decoders typically include a buffer that stores 241,664 bytes of a compressed video data.

12. MPEG-1 compressed video data is typically transmitted at a frame rate of 30 frames/second, a bit-rate of 150 Kbytes/second and fourteen frames per GOP. Thus, an MPEG-1 GOP typically includes approximately 70 Kbytes of compressed video data.

13. MPEG-2 compressed video data is typically transmitted at a frame rate of 30 frames/second, a bit-rate of 750 Kbytes/second and fourteen frames per GOP. Thus, an MPEG-2 GOP typically includes approximately 350 Kbytes of compressed video data.

14. If a still image were encoded in accordance with the Gordon patent's disclosure quoted above in paragraph 9 rather than

in accordance with the disclosure of the present application quoted above in paragraph 10, then an I-frame encoded in accordance with the disclosure of the Gordon patent would overflow buffers respectively included in typical MPEG-1 and MPEG-2 decoders.

15. Typically, a decoder error occurs if compressed video data for an I-frame exceeds the storage capacity of the decoder's buffer.

16. Typically, a decoder error due to buffer overflow prevents presenting an image from compressed video data which overflowes the decoder's buffer.

17. Consequently, for typical transmission of MPEG compressed video data and for typical MPEG decoders, nothing will appear of a still image encoded in accordance with the disclosure of the Gordon patent.

18. Based upon my analysis, explained more specifically below, I conclude that decoding an elementary MPEG bit stream compiled by at least one embodiment disclosed the Gordon patent produces images that pulse visually.

19. The invention disclosed in my patent application produces an elementary MPEG bit stream by combining:

a. headers;

b. an I frame;

c. null P frame; and

d. stuffing.

20. Conversely, the Gordon patent discloses producing an elementary MPEG bit stream by repeating GOPs.

21. In addition to the omissions of the Gordon patent listed above and the differences between the invention disclosed and claimed in my patent application and the disclosure in the Gordon patent, I have identified the following erroneous statements in the disclosure of the Gordon patent.

a. In FIG. 3 and in a description thereof set forth in col. 5 at line 59-60, the Gordon patent incorrectly expressly discloses that a target bit rate is determined in step 310. This is erroneous because the target bit rate must be determined during system specification, i.e. when configuring both the MPEG encoder and the target MPEG decoder.

b. Decoding of an elementary MPEG bit stream generated by the predictive loop, which the Gordon patent depicts in elements 327,-329 of FIG. 3 and the text in col. 6 at lines 37-50, produces images that pulse visually for the reasons explained in the patent application identified above beginning on page 8 at line 22.

c. The Gordon patent states in col. 4 at lines 58-63 that:

> [i]n the case of an MPEG2 information stream, a NULL frame utilized by the inventor comprises a 38 byte data structure that informs the decoder to utilize all the macroblocks from the previous anchor frame and to do so without displacing the macroblocks (i.e., zero motion vectors).

MPEG-2 cannot escape past a slice, and a slice can not extend past a horizontal scan line. Therefore, the preceding statement is incorrect because the minimum amount of data required to encode a single 720×480 frame in accordance with the MPEG-2 specification is 318 bytes, not 38 bytes as disclosed in the text quoted above from the Gordon patent.

d. From col. 6, line 62 through col. 7 line 4 the Gordon patent states that:

> data representative of the stored GOP is coupled to the output of GOP replicator 120 as video elementary stream CV. After replicating the stored GOP (step 340) the routine 300 optionally proceeds to step 345, where time stamps within the individual frames comprising the group of pictures are adjusted. For example, the presentation time stamp (PTS) and/or decode time stamp (DTS) associated with each of the frames may be adjusted to provide appropriate timing information to the resulting bitstream.

The preceding statement is incorrect because the presentation time stamp (PTS) and/or decode time stamp (DTS) are part of an MPEG system stream and are not part of an MPEG elementary stream.

22. During an interval beginning February 13, 1996, and ending April 24, 1998, I was employed as a consultant collaborating with Donald F. Gordon in connection with developing products to be marketed by Diva Systems Corporation ("Diva"), the assignee of the Gordon patent.

23. After I was no longer employed as a consultant by Diva, I developed the invention disclosed and claimed in the patent application identified above.

24. Attached hereto as Exhibit A is a printed document which reproduces a file that:

a. is recorded on a CD-ROM; and

b. bears a date stamp of June 4, 1998.

25. Exhibit A lists a computer program that I wrote on or about June 4, 1998, which implements the invention disclosed and claimed in the patent application identified above.

26. For example, a computer program text appearing in the lower half of page 3 of Exhibit A and continuing onto the second line on page 4 is substantially identical to a computer program text that appears on page 27 of the present patent application.

27. Similarly, a computer program text appearing in the lower half of page 4 of Exhibit A and continuing onto the second line on page 5 is substantially identical to a computer program text that appears on page 28 of the present patent application.

28. Exhibit A establishes that at least by June 4, 1998, I had:

a. thought of the invention disclosed and claimed in the patent application identified above; and

b. written a computer program which implemented the invention.

29. Since it is my custom after writing a computer program to test it by compiling the computer program on a computer which I use for such purposes that is located in the State of California, and then executing the compiled computer program on that same computer, Exhibit A further establishes that on or about June 4, 1998, following my usual practice, the computer program appearing in

Exhibit A would have been compiled and executed within the State of California.

30. I am unaware of any facts contrary to the facts and opinions contained in this Declaration.

31. I declare under penalty of perjury under the laws of the United States of America that all statements made herein of my own knowledge are true and correct, and that all statements made on information and belief are believed to be true; and further that these statements were made with the knowledge that willful false statements and the like so made are punishable by fine or imprisonment, or both under Section 1001 of Title 18 of the United States Code and that such willful false statements may jeopardize the validity of any patent issuing on the subject application.

[illegible]

Mark D. Conover

Dated: 12 July, 2002

```
#define USEMPEG1

/*  Project encode
    SUBSYSTEM:     Encode Object
    FILE:          znullp.cpp
    AUTHOR:        Mark D Conover


    OVERVIEW
    ========
    Source file for implementation null b frame.
*/

/***********************************************************************/
/***********************************************************************/
/*
/*
/*
/* PROPRIETARY RIGHTS NOTICE
/*
/*    This material contains valuable proprietary and trade secret
/*    information of PixelTools of Cupertino, California.  Except in
/*    furtherance of the business activities of PixelTools, Inc. no part
/*    of such information may be disclosed, used, reproduced, or
/*    transmitted in any form or by any means -- electronic, mechanical,
/*    optical or otherwise including photocopying and recording
/*    in connection with any information processing, storage or
/*    retrieval system -- without prior written permission from
/*    PixelTools, Inc.
/*
/*                  ****************************
/*
/* COPYRIGHT NOTICE
/*
/*    Copyright 1995 - 1998
/*    PixelTools Inc
/*    10721 Wunderlich Drive
/*    Cupertino, California 95014
/*    (408) 374-5327
/*    FAX (408) 374-8074
/*
/*    All worldwide rights reserved
/*
/*
/***********************************************************************/
/***********************************************************************/


#include "mpeg2.h"
#include "zencode.h"


/// table also in zputvlc.cpp
/* type definitions for variable length code table entries */

static VLCtable addrinctab[33]=
{
  {0x01,1},  {0x03,3},  {0x02,3},  {0x03,4},
  {0x02,4},  {0x03,5},  {0x02,5},  {0x07,7},
  {0x06,7},  {0x0b,8},  {0x0a,8},  {0x09,8},
  {0x08,8},  {0x07,8},  {0x06,8},  {0x17,10},
  {0x16,10}, {0x15,10}, {0x14,10}, {0x13,10},
  {0x12,10}, {0x23,11}, {0x22,11}, {0x21,11},
  {0x20,11}, {0x1f,11}, {0x1e,11}, {0x1d,11},
  {0x1c,11}, {0x1b,11}, {0x1a,11}, {0x19,11},
  {0x18,11}
};


extern void EncFrameStatsBits(char *text_out); // print out if
// Frame Stats Bits are opened


int MP2Enc::AddNullPFrames(int NumPFrames, int TempRef, int VBVDelay)
{
   ExC.NullPFramesInGop += NumPFrames; /// keep record for correct temp_ref

    TempRef +=  ExC.NullPFramesInGop;
if (tce.mpeg1)
   {
   PutMpeg1Frame(NumPFrames,TempRef,VBVDelay );
   }
else // mpeg2
   {
   if (tce.fieldpic)
      {
      PutMpeg2Field(NumPFrames,TempRef,VBVDelay);
      }
    else
      {
      PutMpeg2Frame(NumPFrames,TempRef,VBVDelay);
      }

   }


  alignbits();



   return 0;
}

int MP2Enc::AddNullBFrames(int NumBFrames, int TempRef, int VBVDelay)
{
int fr,iii;
//int TotalBits;
int TotalBlocks = (tce.width *tce.height)/(16*16);
int NumEscapes = (TotalBlocks - 2)/33;
int NumRemainingBlocks = TotalBlocks - NumEscapes*33 - 2;


double NullBytes = ExC.Floatbytecnt;
  alignbits();

 return 0;
}


int  MP2Enc::ReadLastFrame(char *,unsigned int )
{

/// addd rate control
return 0;
}
```

```
int  MP2Enc::WriteStoredFrame(int ,int ,int)
{


   fwrite (ExC.IBuffer,1,ExC.IPoint,tce.outfile);
   ExC.bytecnt += ExC.IPoint;
   ExC.Floatbytecnt += ExC.IPoint;


return 0;
}



void MP2Enc::CheckForStuffing(void)
{


calc_vbv_delay(); /// compute delay and buffer status
StuffBits();   // add bits if necessary

}


int MP2Enc::PutMpeg1Frame (int NumPFrames, int TempRef, int VBVDelay)
{
int fr,iii;
int TotalBlocks = (tce.width *tce.height)/(16*16);
int NumEscapes = (TotalBlocks - 2)/33;
int NumRemainingBlocks = TotalBlocks - NumEscapes*33 - 2;
double NullBytes = ExC.Floatbytecnt;
  alignbits();

for(fr = 0; fr < NumPFrames; fr+=1)
   {
      putbits (PICTURE_START_CODE,32); /// header
      putbits (TempRef+fr,10);
      putbits (P_TYPE,3);
      putbits (VBVDelay,16);
      putbits (0x0,1); //for full_pel_forward_code
      putbits (0x1,3); // forward_f_code           5
      putbits (0x0,7);   //// stuffing so start code aligns
      putbits (SLICE_MIN_START,32); // slice start
      putbits (0x1,5);        /// m quant
      putbits (0x0,1);       //  extra_slice_bat neglected by MPEG-1 example!!
      putbits (0x1,1);        // macroblock increment (1)

      putbits (0x1,3);        ///macroblock type   (hor_forward)
      putbits (0x1,1);        // horizontal change (0) from 0 to 1 in example)
      putbits (0x1,1);        // vertical   change (0) from 0 to 1

      for (iii = 0; iii < NumEscapes; iii+=1)
         putbits (0x8,11); ///escape     9  * 33 MacroBlocks

     putbits(addrinctab[NumRemainingBlocks].code,addrinctab[NumRemainingBlocks].len);

      putbits (0x1,3);   // type
      putbits (0x1,1);   // change from 0 to 1
      putbits (0x1,1);   // change from 0 to 1
      putbits (0x0,1);   // one less than spec42   ==5+   33+ = 264+2    -- 27
      putbits (0x0,32) ; // flushbytes


/// addd rate control
  alignbits();

 NullBytes = ExC.Floatbytecnt - NullBytes;
   ExC.bytecnt += (int) NullBytes;
   ExC.Floatbytecnt += NullBytes;

   char text_out[256];
   sprintf (text_out,"  Mpegl Frame Null P Frame Bytes Added:%d TempRef::%d\r\n",
   (int) NullBytes, TempRef+fr);
    EncFrameStatsBits(text_out);

    }   /// each frame loop

}



int MP2Enc::PutMpeg2Field (int NumPFrames, int TempRef, int VBVDelay)
{
int fr,iii;
int TotalBlocks = (tce.width *tce.height)/(16*16*2); //*2 for each field
int NumEscapes = (TotalBlocks - 2)/33;
int NumRemainingBlocks = TotalBlocks - NumEscapes*33 - 2;
double NullBytes = ExC.Floatbytecnt;
  alignbits();


for(fr = 0; fr < NumPFrames; fr+=1)
   {

/// Field Top

      putbits (PICTURE_START_CODE,32); /// header
      putbits (TempRef+fr,10);
      putbits (P_TYPE,3);
      putbits (VBVDelay,16);
      putbits (0x0,1); //for full_pel_forward_code
// Mpegl      putbits (0x1,3); // forward_f_code           5
      putbits (0x7,3); // forward_f_code   //MPeg2         5

//   MPEG1   putbits (0x0,?);   //// stuffing so start code aligns
      putbits (0x0,7);   //// stuffing so start code aligns
      AddNullPictureExtension(1);  // top field
      putbits (SLICE_MIN_START,32); // slice start
      putbits (0x1,5);         /// m quant
      putbits (0x0,1);         //  extra_slice_bat neglected by MPEG-1 example!!
      putbits (0x1,1);         // macroblock increment (1)
                  //     macroblock_modes()
      putbits (0x1,3);         ///macroblock type   (hor_forward) macroblock_motion_forward (Mpeg2)
 // M1    putbits (0x1,1);        // horizontal change (0) from 0 to 1 in example)
//  M1    putbits (0x1,1);        // vertical   change (0) from 0 to 1


      putbits (0x1,2);         // field motion
      putbits (0x1,1);         // motion_vertical_field_select[]
      putbits (0x1,1);         // Motion vector of 0
      putbits (0x1,1);         // Second Motion vector of 0
///      putbits (0x1,1);        // Marker bit not in Field Pics!
                                                  /// 15 bits since MIN_START

// now add macro_block excapes
```

```
      for (iii = 0; iii < NumEscapes; iii+=1)
         putbits (0x8,11); ///escape     9  * 33 MacroBlocks  .


//     MB increment for last macro-block
     putbits(addrinctab[NumRemainingBlocks].code,addrinctab[NumRemainingBlocks].len);

      putbits (0x1,3);   // type

// m1      putbits (0x1,1);   // change from 0 to 1
 // m1      putbits (0x1,1);   // change from 0 to 1

      putbits (0x1,2);         // field motion
      putbits (0x1,1);         // motion_vertical_field_select[]
      putbits (0x1,1);         // Motion vector of 0
      putbits (0x1,1);         // Second Motion vector of 0
///      putbits (0x1,1);         // Marker bit not in Field Pics!


//      putbits (0x0,1);   // one less than spec42    ==5+   33+ = 264+2    -- 27


      putbits (0x0,32) ; // flushbytes out of pipe


/// addd rate control
   alignbits();  // add bytes to align to boundaries



/// Field Bottom

      putbits (PICTURE_START_CODE,32); /// header
      putbits (TempRef,10);
      putbits (P_TYPE,3);
      putbits (VBVDelay,16);
      putbits (0x0,1); //for full_pel_forward_code
// Mpeg1      putbits (0x1,3); // forward_f_code          5
      putbits (0x7,3); // forward_f_code   //MPeg2          5

//   MPEG1   putbits (0x0,?);   //// stuffing so start code aligns
      putbits (0x0,7);   //// stuffing so start code aligns
      AddNullPictureExtension(2);  // bottom field

      putbits (SLICE_MIN_START,32); // slice start
      putbits (0x1,5);         /// m quant
      putbits (0x0,1);         //  extra_slice_bat neglected by MPEG-1 example!!
      putbits (0x1,1);         // macroblock increment (1)
               //     macroblock_modes()
      putbits (0x1,3);         ///macroblock type   (hor_forward) macroblock_motion_forward (Mpeg2)
 // M1    putbits (0x1,1);        // horizontal change (0) from 0 to 1 in example)
//  M1    putbits (0x1,1);        // vertical   change (0) from 0 to 1


      putbits (0x1,2);         // field motion
      putbits (0x1,1);         // motion_vertical_field_select[]
      putbits (0x1,1);         // Motion vector of 0
      putbits (0x1,1);         // Second Motion vector of 0
///      putbits (0x1,1);         // Marker bit not in Field Pics!
                                         /// 15 bits since MIN_START

// now add macro_block excapes
      for (iii = 0; iii < NumEscapes; iii+=1)
         putbits (0x8,11); ///escape     9  * 33 MacroBlocks
```

```
//     MB increment for last macro-block
      putbits(addrinctab[NumRemainingBlocks].code,addrinctab[NumRemainingBlocks].len);

       putbits (0x1,3);   // type

// ml      putbits (0x1,1);   // change from 0 to 1
 // ml      putbits (0x1,1);   // change from 0 to 1

       putbits (0x1,2);         // field motion
       putbits (0x1,1);         // motion_vertical_field_select[]
       putbits (0x1,1);         // Motion vector of 0
       putbits (0x1,1);         // Second Motion vector of 0
///      putbits (0x1,1);         // Marker bit not in Field Pics!


//      putbits (0x0,1);   // one less than spec42    ==5+    33+ = 264+2    -- 27


       putbits (0x0,32) ; // flushbytes out of pipe


/// addd rate control
   alignbits();  // add bytes to align to boundaries



 NullBytes = ExC.Floatbytecnt - NullBytes;
   ExC.bytecnt += (int) NullBytes;
   ExC.Floatbytecnt += NullBytes;
  char text_out[256];
   sprintf (text_out,"   Mpeg2 Field Null P Frame Bytes Added:%d TempRef::%d\r\n",
   (int) NullBytes, TempRef+fr);
    EncFrameStatsBits(text_out);

     }   /// each frame loop


}


void MP2Enc::AddNullPictureExtension(int Null_picture_structure)
// picture_structure = 1 top field; 2 bottom field; 3 frame picture
{
  alignbits();
  putbits(EXT_START_CODE,32); /* extension_start_code */
  putbits(CODING_ID,4); /* extension_start_code_identifier */
  putbits(tce.forw_hor_f_code,4); /* forward_horizontal_f_code */
  putbits(tce.forw_vert_f_code,4); /* forward_vertical_f_code */
  putbits(tce.back_hor_f_code,4); /* backward_horizontal_f_code */
  putbits(tce.back_vert_f_code,4); /* backward_vertical_f_code */
  putbits(tce.dc_prec,2); /* intra_dc_precision */
  putbits(Null_picture_structure,2); /* picture_structure */
  putbits((tce.pict_struct==FRAME_PICTURE)?tce.topfirst:0,1); /* top_field_first */
  putbits(tce.frame_pred_dct,1); /* frame_pred_frame_dct */
  putbits(0,1); /* concealment_motion_vectors  -- currently not implemented */
  putbits(tce.q_scale_type,1); /* q_scale_type */
//  putbits(tce.intravlc,1); /* intra_vlc_format */
  putbits(0,1); /* force it to 0 */
  putbits(tce.altscan,1); /* alternate_scan */
  putbits(tce.repeatfirst,1); /* repeat_first_field */
  putbits(tce.prog_frame,1); /* chroma_420_type */
  putbits(tce.prog_frame,1); /* progressive_frame */
  putbits(0,1); /* composite_display_flag */
}
```

```
#ifndef USEMPEG1
/// try wiht mpeg2 pixture extensions
int MP2Enc::PutMpeg2Frame (int NumPFrames, int TempRef, int VBVDelay)
{
int fr,iii;
int TotalBlocks = (tce.width *tce.height)/(16*16*2); //*2 for each field
int NumEscapes = (TotalBlocks - 2)/33;
int NumRemainingBlocks = TotalBlocks - NumEscapes*33 - 2;
double NullBytes = ExC.Floatbytecnt;
  alignbits();


for(fr = 0; fr < NumPFrames; fr+=1)
   {
      putbits (PICTURE_START_CODE,32); /// header
      putbits (TempRef+fr,10);
      putbits (P_TYPE,3);
      putbits (VBVDelay,16);
      putbits (0x0,1); //for full_pel_forward_code
// Mpeg1      putbits (0x1,3); // forward_f_code          5
      putbits (0x7,3); // forward_f_code   //MPeg2         5

//   MPEG1   putbits (0x0,?);   //// stuffing so start code aligns
      putbits (0x0,7);   //// stuffing so start code aligns
      AddNullPictureExtension(3); // frame picture
      putbits (SLICE_MIN_START,32); // slice start
      putbits (0x1,5);          /// m quant
      putbits (0x0,1);          //  extra_slice_bat neglected by MPEG-1 example!!
      putbits (0x1,1);          // macroblock increment (1)
                 //     macroblock_modes()
      putbits (0x1,3);          ///macroblock type   (hor_forward) macroblock_motion_forward (Mpeg2)
 // M1    putbits (0x1,1);        // horizontal change (0) from 0 to 1 in example)
//  M1    putbits (0x1,1);        // vertical   change (0) from 0 to 1


      putbits (0x2,2);          // frame motion
      putbits (0x1,1);          // motion_vertical_field_select[]
      putbits (0x1,1);          // Motion vector of 0
      putbits (0x1,1);          // Second Motion vector of 0

      putbits (0x1,1);          // motion_vertical_field_select[] /// 4 vectores for frame
      putbits (0x1,1);          // Motion vector of 0
      putbits (0x1,1);          // Second Motion vector of 0



///       putbits (0x1,1);        // Marker bit not in Field Pics!
                                             /// 15 bits since MIN_START

// now add macro_block excapes
      for (iii = 0; iii < NumEscapes; iii+=1)
         putbits (0x8,11); ///escape     9  * 33 MacroBlocks


//     MB increment for last macro-block
     putbits(addrinctab[NumRemainingBlocks].code,addrinctab[NumRemainingBlocks].len);

      putbits (0x1,3);   // type

// m1      putbits (0x1,1);   // change from 0 to 1
 // m1      putbits (0x1,1);   // change from 0 to 1

      putbits (0x2,2);         // frame motion
      putbits (0x1,1);         // motion_vertical_field_select[]
      putbits (0x1,1);         // Motion vector of 0
      putbits (0x1,1);         // Second Motion vector of 0

      putbits (0x1,1);         // motion_vertical_field_select[] /// 4 vectores for frame
      putbits (0x1,1);         // Motion vector of 0
      putbits (0x1,1);         // Second Motion vector of 0

///      putbits (0x1,1);        // Marker bit not in Field Pics!


//     putbits (0x0,1);   // one less than spec42    ==5+   33+ = 264+2    -- 27


      putbits (0x0,32) ; // flushbytes out of pipe


/// addd rate control
  alignbits(); // add bytes to align to boundaries




 NullBytes = ExC.Floatbytecnt - NullBytes;
   ExC.bytecnt += (int) NullBytes;
   ExC.Floatbytecnt += NullBytes;

   char text_out[400];
   sprintf (text_out,"  Mpeg2 Field Null P Frame Bytes Added:%d TempRef::%d\r\n",
   (int) NullBytes, TempRef+fr);
    EncFrameStatsBits(text_out);


     }   /// each frame loop
}

#else
// works like mpeg-1
int MP2Enc::PutMpeg2Frame (int NumPFrames, int TempRef, int VBVDelay)
{
int fr,iii;
int TotalBlocks = (tce.width *tce.height)/(16*16);
int NumEscapes = (TotalBlocks - 2)/33;
int NumRemainingBlocks = TotalBlocks - NumEscapes*33 - 2;
double NullBytes = ExC.Floatbytecnt;
  alignbits();

for(fr = 0; fr < NumPFrames; fr+=1)
   {
      putbits (PICTURE_START_CODE,32); /// header
      putbits (TempRef,10);
      putbits (P_TYPE,3);
      putbits (VBVDelay,16);
      putbits (0x0,1); //for full_pel_forward_code
      putbits (0x1,3); // forward_f_code          5
      putbits (0x0,7);   //// stuffing so start code aligns
      putbits (SLICE_MIN_START,32); // slice start
      putbits (0x1,5);         /// m quant
      putbits (0x0,1);        //  extra_slice_bat neglected by MPEG-1 example!!
      putbits (0x1,1);         // macroblock increment (1)
      putbits (0x1,3);         ///macroblock type   (hor_forward)
      putbits (0x1,1);         // horizontal change (0) from 0 to 1 in example)
      putbits (0x1,1);         // vertical   change (0) from 0 to 1
```

```
       for (iii = 0; iii < NumEscapes; iii+=1)
          putbits (0x8,11); ///escape     9  * 33 MacroBlocks

      putbits(addrinctab[NumRemainingBlocks].code,addrinctab[NumRemainingBlocks].len);

       putbits (0x1,3);    // type
       putbits (0x1,1);    // change from 0 to 1
       putbits (0x1,1);    // change from 0 to 1
       putbits (0x0,1);    // one less than spec42    ==5+   33+ = 264+2    -- 27
       putbits (0x0,32) ; // flushbytes


/// addd rate control
  alignbits();

 NullBytes = ExC.Floatbytecnt - NullBytes;
   ExC.bytecnt += (int) NullBytes;
   ExC.Floatbytecnt += NullBytes;
      char text_out[256];
   sprintf (text_out,"  Mpeg2 Frame Null P Frame Bytes Added:%d TempRef::%d\r\n",
   (int) NullBytes, TempRef+fr);
    EncFrameStatsBits(text_out);

     }   /// each frame loop

}  //end NullPFrame



#endif // USEMPEG1
```